

```

/* lex.h

Copyright (c) 1993-2008. Free Software Foundation, Inc.

This file is part of GNU MCSim.

GNU MCSim is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 3
of the License, or (at your option) any later version.

GNU MCSim is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with GNU MCSim; if not, see <http://www.gnu.org/licenses/>

-- Revisions -----
Logfile: %F%
Revision: %I%
Date: %G%
Modtime: %U%
Author: @a
-- SCCS -----

Header file for Lexical parsing routines.
*/
#ifndef LEX_H_DEFINED

/* -----
-----
Inclusions */

#include "hungtype.h"

/* -----
-----
Constants */

#define BUFFER_SIZE      0x1000      /* Size of input data buffer */
#define MAX_LEX          0x03FF      /* Max size of Lexical Element */
#define MAX_EQN          0x03FF      /* Max size of a string eqn */
#define MAX_NAME         80          /* Max size of a name */

/* -----
-----
Lexical types */

#define LX_NULL          0x0000

```

```

#define LX_IDENTIFIER 0x0001
#define LX_INTEGER    0x0002
#define LX_FLOAT      0x0004
#define LX_NUMBER     (LX_INTEGER | LX_FLOAT)
#define LX_PUNCT      0x0008
#define LX_STRING     0x0010
#define LX_EQNPUNCT   0x0020

/* To avoid unmatched delimiters confusions in editor */

#define CH_LPAREN      '('
#define CH_RPAREN      ')'
#define CH_LBRACKET    '['
#define CH_RBRACKET    ']'
#define CH_LBRACE      '{'
#define CH_RBRACE      '}'


/* Character constants for convenience */

#define CH_EOLN        ('\n') /* End of line character */
#define CH_COMMENT     ('#') /* One line Comment Char */
#define CH_STRDELIM    ('\"') /* String delimiter */
#define CH_STMTTERM    (';') /* Statement terminator */


/* Report Error constants -- Lex errors */

#define MAX_ERRORS 20

#define RE_FATAL       0x8000 /* Can be OR'd to iCode to cause exit(1) */
#define RE_WARNING     0x4000 /* can be OR'd to issue Warning instead */

#define RE_UNKNOWN     0x0000 /* Unspecified error */
#define RE_INIT        0x0001 /* Error during initialization */
#define RE_FILENOFOUND 0x0002 /* Error opening file for I/O */
#define RE_CANNOTOPEN   0x0003 /* Cannot open file */
#define RE_OUTOFMEM    0x0004 /* Error allocating memory */

#define RE_UNEXPECTED  0x0011 /* Unexpected char in input */
#define RE_UNEXPNUMBER 0x0012 /* Unexpected number in input */
#define RE_EXPECTED    0x0013 /* Expected character szMsg[0] */
#define RE_LEXEXPECTED 0x0014 /* Expected szMsg lexical element */

#define RE_SBWERROR    0x0015 /* System Biology Workbench error */
#define RE_SBWNOMERROR 0x0016 /* System Biology Workbench NOM service
error */

/* User defined errors starting with 0x0100 */

#define RE_USERERROR   0x0100 /* User Error prefix */

/* Model Generator Errors */

#define RE_MODERROR    0x0100 /* Model Generator error prefix */

```

```

#define RE_BADCONTEXT      (RE_MODERROR + 1) /* Invalid context for ident */
#define RE_DUPDECL         (RE_MODERROR + 2) /* Duplicate declaration */
#define RE_REDEF            (RE_MODERROR + 3) /* Redefinition of parm */
#define RE_EQNTOOLONG       (RE_MODERROR + 4) /* Equation too long for
buffer */
#define RE_BADSTATE         (RE_MODERROR + 5) /* Invalid state id */
#define RE_UNDEFINED         (RE_MODERROR + 6) /* Undefined identifier */
#define RE_NODYNEQN          (RE_MODERROR + 7) /* Missing dyn eqn for szMsg
*/
#define RE_NOINPDEF          (RE_MODERROR + 8) /* Input not init'd */
#define RE_TOOMANYVARS        (RE_MODERROR + 9) /* Too many variable decls */
#define RE_POSITIVE           (RE_MODERROR + 10) /* Positive number expected */
#define RE_NAMETOOLONG        (RE_MODERROR + 11) /* Variable name too long */
#define RE_UNBALPAR           (RE_MODERROR + 12) /* Unbalanced parentheses */
#define RE_NOOUTPUTEQN         (RE_MODERROR + 13) /* Missing dyn eqn for szMsg
*/
#define RE_DUPSECT            (RE_MODERROR + 14) /* Duplicated section szMsg */
#define RE_NOEND              (RE_MODERROR + 15) /* Missing End keyword */

#define RE_SIMERROR           0x0200 /* Simulation error prefix */

/* -----
-----
 Public Typedefs */

typedef PSTR PBUF;

/* The INPUTBUF structure which is used for file I/O buffering */
typedef struct tagINPUTBUF {
    PFILE pfileIn;      /* file pointer */
    PBUF pbufOrg;       /* Pointers for buffer Origin */
    long lBufSize;       /* Size of pbufOrg */
    PBUF pbufCur;       /* ... Current point */
    int iLineNum;        /* Line number in file */
    int iLNPrev;         /* Prev line num. For formatting Dynamics eqns */
    int cErrors;         /* Count of Errors */

    PVOID pInfo;         /* Pointer to private user information */
    PVOID pTempInfo;     /* Pointer to private user template information */
} INPUTBUF, * PINPUTBUF;

typedef char PSTRLEX[MAX_LEX]; /* String of a lexical element */
typedef char PSTREQN[MAX_EQN]; /* String of an equation */

/* -----
-----
 Public Macros */

#define EOB(pib) ((!(pib)) \
|| ((!(pib))->pbufCur || !*(pib)->pbufCur) \

```

```

    && (! (pib) ->pfileIn || feof((pib) ->pfileIn)))

#define IsUnderscore(c)      ((c) == '_')
#define IsSign(c)           ((c) == '+' || (c) == '-')
#define IsComment(szLex)    ((szLex) ? (* (szLex) == CH_COMMENT) : (0) )
#define IsString(szLex)     ((szLex) ? (* (szLex) == CH_STRDELIM) : (0) )

#define ErrorsReported(pib) ((pib) ->cErrors)
#define ClearErrors(pib)   ((pib) ? (pib) ->cErrors = 0 : 0)

/*
-----
Prototypes */

void EatStatement (PINPUTBUF pib);
int EGetPunct (PINPUTBUF pibIn, PSTR szLex, char chPunct);
int ENextLex (PINPUTBUF, PSTRLEX, int);
long EvalAtom (PINPUTBUF pibIn, long index, PSTR *szExp, PSTR szToken,
               PINT piType);
long EvalParen (PINPUTBUF pibIn, long index, PSTR *szExp, PSTR szToken,
                PINT piType);
long EvalProd (PINPUTBUF pibIn, long index, PSTR *szExp, PSTR szToken,
               PINT piType);
long EvalSum (PINPUTBUF pibIn, long index, PSTR *szExp, PSTR szToken,
               PINT piType);
long EvaluateExpression (PINPUTBUF pibIn, long index, PSTR szExpress);
long EvalUnary (PINPUTBUF pibIn, long index, PSTR *szExp, PSTR szToken,
                PINT piType);

int FillBuffer (PINPUTBUF pibIn, long lBuffer_size);
void FlushBuffer (PINPUTBUF pibIn);

void GetArrayBounds (PINPUTBUF pibIn, PLONG piLB, PLONG piUB);
void GetaString (PINPUTBUF pibIn, PSTR szLex);
BOOL GetFuncArgs (PINPUTBUF pibIn, int nArgs, int rgiArgTypes[], PSTR
szArgs,
                  long rgiLowerB[], long rgiUpperB[]);
void GetIdentifier (PINPUTBUF pibIn, PSTR szLex);
void GetInteger (PINPUTBUF pibIn, PSTR szLex, PINT piLexType);
void GetNumber (PINPUTBUF pibIn, PSTR szLex, PINT piLexType);
int GetOptPunct (PINPUTBUF, PSTR, char);
int GetPunct (PINPUTBUF pibIn, PSTR szLex, char chPunct);
void GetStatement (PINPUTBUF pibIn, PSTR szStmt);
void GetToken (PSTR *szExpress, PSTR szToken, PINT piType);

BOOL InitBuffer (PINPUTBUF pibIn, long lBuffer_size, PSTR
szFullPathname);

void MakeStringBuffer (PINPUTBUF pBuf, PINPUTBUF pStrBuf, PSTR sz);

char NextChar (PINPUTBUF pibIn);
void NextLex (PINPUTBUF, PSTRLEX, PINT);
int NextListItem (PINPUTBUF, PSTR, int, int, char);

```

```
void PreventLexSplit (PINPUTBUF pibIn, int iOffset);

void SkipComment (PINPUTBUF);
int SkipWhitespace (PINPUTBUF pibIn);

void UnrollEquation (PINPUTBUF pibIn, long index, PSTR szEqn, PSTR
szEqnU);

#define LEX_H_DEFINED
#endif

/* End */
```